REMARKS

Claims 1-13 and 15-19 have been rejected under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 6,230,316 to Nachenberg ("Nachenberg") in view of U.S. Patent Pub. No. 2002/0026634 to Shaw ("Shaw").

Claim 14 has been previously canceled.

Claims 1-13 and 15-19 remain pending.

Rejection of Claims 1-13 and 15-19 under 35 U.S.C. §103(a)

With respect to independent claim 1, the Office Action states that Nachenberg teaches all of Applicants' recited elements except signed pieces of code, which Shaw allegedly teaches.

Nachenberg discloses incremental updating of an executable file that has been rebased or realigned. Nachenberg teaches that when an incremental software update is distributed, it may be rebased, which means memory addresses that appear in code and data segments are changed to accommodate the file being loaded into memory at a new base address. Furthermore, the software update may be realigned, which refers to moving the code and data segments within a file (see col. 1, line 66 to col. 2, line 10). Nachenberg states that such rebasing and realigning can result in unpredictable results when a software update occurs by binary patching. Binary patching is a technique for incrementally updating software in which the bits of the software that are different in a new version of the software are changed (see col. 1, lines 39-48, and lines 60-65).

The Examiner cites Figure 8 and col. 5, line 36 to col. 6, line 17 of Nachenberg as teaching combining the difference code with a first signed piece of code by a second software archive generator to generate the second signed piece of code, whereby the second software

archive generator is fed with those generation instructions that were used by the first software archive generator for the generation of both pieces of code. Applicants submit that the cited passages have been misinterpreted.

The passages cited at col. 5, line 36 to col. 6, line 17 of Nachenberg describe avoiding the unpredictable results that may occur due to rebasing or realigning by converting application files to a canonical form before performing an update. For example, as shown in Fig. 8, a version 100A of an application file is converted into a canonical version 100B. An update builder uses a conventional binary patch file builder to determine a binary difference in an update file 124C-B, between the version 100B file and the new version 100C file. The update file 124C-B is then distributed to a user who installs it on the user's computer, which also includes the canonical converter. A pre-update version of the file 100 on the user's computer has been rebased and realigned, such that it is in state 100D. The canonical converter on the user's computer converts the file 100D to the canonical version 100B. The update file 124C-B is then used by an updater to produce the new version 100C of the file.

The updater can be any conventional binary patcher that applies the patch, e.g., update file 124C-B, to the canonical version 100B (see col. 5, lines 35-60 of Nachenberg). Thus, the method disclosed by Nachenberg requires that an additional, intermediate file (canonical version) on both the server end and the user end be created before an update is applied. Nachenberg provides a conventional binary patch file based on the difference between an updated file and a canonical version of a pre-update file. At the user's location, the binary patch file is applied to a canonical version of the pre-update file to derive the updated file.

In contrast, Applicants' invention involves providing information to an end user to allow the user to update a first signed code to a signed second code. This is achieved by providing

generation instructions and a difference code to the user. The generation instructions are instructions that are used by a first software archive generator to generate both the first and second signed code at the location of a software provider. The difference code includes the steps necessary to arrive at the second signed code from the first signed code. The user employs a second software archive generator, which uses the difference code and the first signed code to generate the second signed code based on the generation instructions. Applicants' invention does not require creating an intermediate, canonical version of the code before an update is applied.

Further, Nachenberg does not teach or suggest the use of a second software archive generator (on the user's side) that uses generation instructions that were used by a first software archive generator (on the server side) for generating both the first and second pieces of code. There is no teaching in Nachenberg that the update file 100C is generated from the pre-update file 100A using generation instructions. Instead, it is only assumed that the update file 100C is somehow available. Nachenberg is concerned with providing a file that represents a binary difference between the file 100C and the canonical version (file 100B) of file 100A. Nachenberg also teaches that the updater uses the update file 124C-B to produce version 100C. However, the updater is any conventional binary patcher, which can apply patches. The updater is clearly not the same as Applicants' second software archive generator, which uses the generation instructions that were used by the first software archive generator to generate both the first and second pieces of code.

Shaw discloses a method and system for secure downloading, recovery, and upgrading of data. A client device receives information from a server device using reliable software modules stored in permanent memory in the client device. The reliable software modules perform

software and data integrity tests, and locate and retrieve data for recovery or upgrade of the client device. The client device confirms the trustworthiness of the received information device by comparing digital signatures or digests for the information it receives with known digital certificates in the reliable software module.

Shaw does not teach or suggest a computer-implemented method for a software provider of enabling a software-acquiring entity to arrive from an existent first signed piece of code executable on a machine at a second signed piece of code executable on the machine, where both pieces of code have been generated by use of a first software archive generator under use of generation instructions, as recited in Applicants' claim 1. Specifically, Shaw does not teach or suggest providing to the software-acquiring entity a difference code, where the difference code includes the steps necessary to arrive from the first signed piece of code at the second signed piece of code, where the difference code is usable at the software-acquiring entity. Further, Shaw does not teach or suggest combining the difference code with the first signed piece of code by a second software archive generator to generate the second signed piece of code, whereby the second software archive generator is fed with those generation instructions that were used by the first software archive generator for the generation of both pieces of code.

In view of the foregoing, it is respectfully submitted that Nachenberg and Shaw, whether taken alone or in combination, do not teach or suggest the subject matter recited in Applicants' independent claim 1 as each of these references fails to teach or suggest a computer-implemented method for a software provider of enabling a software-acquiring entity to arrive from an existent first signed piece of code executable on a machine at a second signed piece of code executable on the machine, where both pieces of code have been generated by use of a first software archive generator under use of generation instructions. The method includes providing to the software-

acquiring entity a difference code, where the difference code includes the steps necessary to arrive from the first signed piece of code at the second signed piece of code, where the difference code is usable at the software-acquiring entity. The method further includes combining the difference code with the first signed piece of code by a second software archive generator to generate the second signed piece of code, whereby the second software archive generator is fed with those generation instructions that were used by the first software archive generator for the generation of both pieces of code.

Independent claims 9, 15, and 18 recite similar features as claim 1, and therefore are patentably distinct over Nachenberg and Shaw for at least the reasons discussed in connection with independent claim 1.

Claims 2-8, 10-13, 16-17, and 19 which depend directly from the independent claims 1, 9, 15, and 18, incorporate all of the limitations the corresponding independent claims and are therefore patentably distinct over Nachenberg and Shaw for at least those reasons provided for claims 1, 9, 15, and 18.

Conclusion

In view of the foregoing, Applicants respectfully request reconsideration, withdrawal of all rejections, and allowance of all pending claims in due course.

Respectfully submitted,

Steven Fischman
Registration No. 34,594

Scully, Scott, Murphy & Presser
400 Garden City Plaza, Suite 300
Garden City, N.Y. 11530
(516) 742-4343

SF/BMM:me

G:\IBM\105\14665\amend\14665.am3.doc